Renault SAS Nissan Peugeot Citroën Automobile	Reference : **0001-4** Version **1.0**	Publication date **2009/01/29** Page 1/17

Standard ECU reprogramming
Part 4 - Boot-loader mechanisms

Comment:
This document describes internal mechanisms of the boot-loader used for the programming procedure.

AUTHORS(S)	
Name : Gilles Michard (Renault SAS) Cédric Meunier (Peugeot Citroën Automobile)	

1 Revision summary

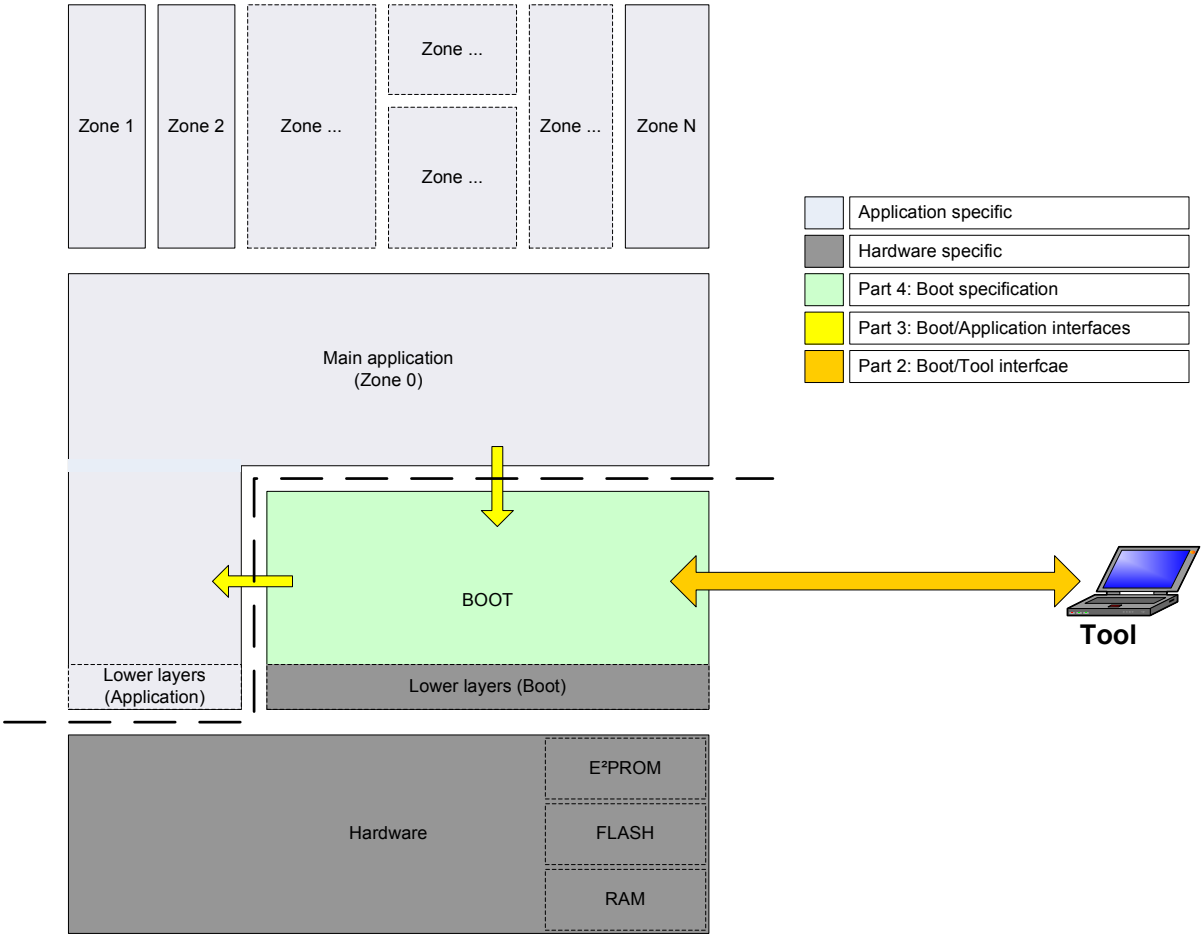
Revision	Date	Modified paragraphs and kind of modification
1.0	January 29th, 2009	First edition

2 Content

3 PURPOSE

This document is a part of Standard ECU reprogramming specification package. This package is divided into five parts, consistent to each others.

- Part 1: General description
- Part 2: Ecu/tool programming interfaces description
- Part 3: Boot-loader/Application interface description
- Part 4: Boot-loader mechanisms
- Part 5: Conformance test



Original documents will be freely available on "Internet Archive" website:
http://www.archive.org/

The Standard ECU reprogramming package contains specification on boot, how it interfere with application (application reprogramming and launch), with tool (how to reprogram an application, conformity test) and means to test the boot.

This part is related to internal boot-loader mechanisms. It specifies how standard boot-loader reacts on its input/output stimulation.

Boot-loader interfaces are:

	Name	**Comment**
Boot/Application		
Boot/Tool	BOOT_PROG_SESSION	
	BOOT_SA_UNLOCK_REQ	
	BOOT_WRITE_DIGEST	
	BOOT_REQ_DOWNLOAD	
	BOOT_TRANSFER_DATA	
	BOOT_RUNDTCTEST	
	BOOT_GET_LOGICALBLOCK_INFO	
Boot/Lower layer		

4 APPLICABLE DOCUMENTS

4.1 References documents

Title	Ref.	Rev.
[1] Standard ECU reprogramming - Part 2 - Ecu-tool programming interfaces description	0001-2	1.0

4.2 Norms and Procedures

Title	Ref.	Rev.
[2] Road vehicles — Unified diagnostic services (UDS) — Part 1: Specification and requirements	ISO 14229-1	2006
[3] Road vehicles — Diagnostics on Controller Area Networks (CAN) — Part 3:Implementation of unified diagnostic services (UDS on CAN)	ISO 15765-3	2004
[4] Road vehicles — Communication between vehicle and external equipment for emissions-related diagnostics — Part 6: Diagnostic trouble code definitions	ISO 15031-6	2005

5 TERMINOLOGY

5.1 Glossary

Segment	:	A segment is a in memory contiguous part of data.
Logical Block	:	A logical block is a block of consistent data that can be unitarily be reprogrammed (e.g. calibration or software module). A logical block is build of one or more segments.
Unlocking fingerprint	:	This is a data that represent the entity for which securityAccess has been allowed..
Digest	:	Result of a cryptographic hash function. It represents the calculated signature of a data set.
Scratchpad	:	Volatile memory part that can be programmed without any security check.
Memory handler	:	Procedure that can access (read/write) to Logical Block

5.2 Abbreviations and acronyms

NRC	:	Negative Response Code. See document [2]

5.3 Conventions

5.3.1 Requirements presentation

Requirements are presented in the form of a table containing following information:
- First column : Requirement identifier (see below the applicable numbering method)
- Second column : Requirement Description

A requirement can be followed by two row, a rationale and a comment that can help the reader to understand the requirement.

STD-PRG4-TS.nnnn(V)	Requirement Description
Rationale	*Rationale on the above requirement.*
Comment	*Comment on the above requirement*

5.3.2 Requirement identifiers

For an easy identification of the requirement scope, the following method has been adopted: STD-DIAG-TS.nnnn(V)
where:

STD	:	For standard requirement
PRG	:	For programming related requirement.
4	:	For part 4 related requirement.
TS	:	For technical specifications
nnnn	:	Requirement Identifier number (from 0000 à 9999)
V	:	Requirement version number

6 REQUIREMENTS

6.1 General

STD-PRG4-TS.0300(1)	The boot-loader must support reset at any time.
Rationale	*A loss of power supply must not cause the loss of the ECU.*
Comment	

STD-PRG4-TS.0301(1)	The boot-loader must contain no other procedures than described into this document to modify data contained in the Logical Blocks.
Rationale	*Avoid to provide backdoors that could be use for ECU hacking.*
Comment	

Code that is able to modify memory must be protected against unexpected execution.

6.2 Boot-loader states

Boot-loader states are either DTC statuses or internals Boot-loader states.

6.2.1 Boot-loader DTCs

STD-PRG4-TS.0001(1)	Each DTC of the Boot-loader must be coded using 3-byte format defined into [4] norm.
Rationale	
Comment	*2 bytes for baseDTC and 1 byte for failure type. 0 for failure type should not be used.* *DTCFormatIdentifier (of ReadDTCInformation sub-function 0x01) must be 0x00* *Notation DTC(baseDTC, failureType) is used to represent the value of a DTC.*

STD-PRG4-TS.0002(1)	Each Logical Block is associated to a DTC(Logical Block ID, Programming failure/Not programmed). This DTC provides information on the validity of the data contained into the logical block. Its ISO 15031-6 failure type is 0x51.
Rationale	*DTC status inform on the validity of the logical bloc.*
Comment	*This subtype is used by the control module to indicate that programming is required.*

STD-PRG4-TS.0003(1)	Some Logical Blocks are associated with DTC(Logical Block ID, system internal failures/General memory failure).. This DTC provides information on the integrity of physical memory. Its ISO 15031-6 failure type is 0x42
Rationale	*-*
Comment	*This subtype is used by the control module to indicate that the ECU is subject to a memory hardware failure.*

STD-PRG4-TS.0004(1)	Each DTC provides the information that it has not been tested since last clear.
Rationale	*-*
Comment	*Each DTC of the Boot-loader must support the status testNotCompletedSinceLastClear-*

STD-PRG4-TS.0005(1)	If applicable, the Boot-loader implements a DTC that gives information on its general failure state.
Rationale	*-*
Comment	*-*

6.2.2 Counters

Counter name	**Comment**
LOGICAL_BLOK_REPROG_CNTR(Logical Block ID)	Each ECU can contain several Logical Blocks. The counter is the number of remaining possible update operations on the Logical Block

STD-PRG4-TS.0010 (1)	In case the Logical Block is contained into a memory with a programming limit, the DTC(Logical Block ID, system internal failures/General memory failure) must be supported. The interface BOOT_GET_LOGICALBLOCK_INFO reports only the counters less than 126. See ref [1].
Rationale	*-According to ISO 14229-1 UDS [2] norm (see table 249), returning 127 indicates a "failed" test. This mean that when the value 126 is reached, no more programming operation must be performed.*
Comment	*- BOOT_GET_LOGICALBLOCK_INFO returns: 126 - LOGICAL_BLOK_REPROG_CNTR(Logical Block ID)*

Note: A Logical Block can be based on several memory types (several segments) with several programming limit. LOGICAL_BLOK_REPROG_CNTR(Logical Block ID) must reflect the lowest limit.

Into the following example, the Logical Block is composed of 3 segments of memory.

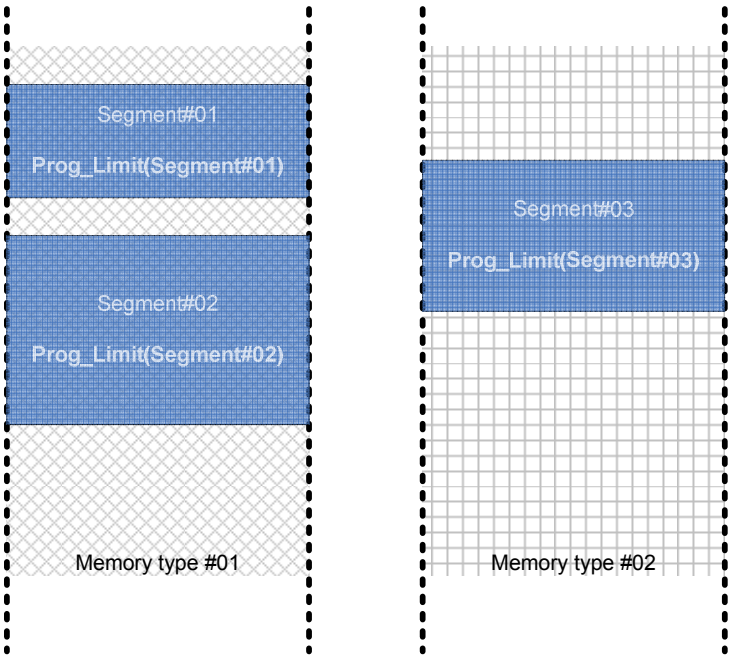


Figure 1 - LOGICAL_BLOK_REPROG_CNTR(Logical Block ID)

In this case, LOGICAL_BLOK_REPROG_CNTR(Logical Block ID) should be the minimum number between Prog_Limit(Segment#01), Prog_Limit(Segment#02) and Prog_Limit(Segment#03)

STD-PRG4-TS.0011(1)	In case the Logical Block is contained into a memory with a programming limit, for each Logical Block, when the LOGICAL_BLOK_REPROG_CNTR(Logical Block ID) reach the value 0, BOOT_REQ_DOWNLOAD is refused.
Rationale	*-*
Comment	*In this case, NRC 0x70 should be used.*

STD-PRG4-TS.0012(1)	In case the Logical Block is contained into a memory without a programming

	limit, LOGICAL_BLOK_REPROG_CNTR(Logical Block ID) is not implemented
Rationale	*-*
Comment	*Case for E²PROM, Hard drives or case where limit is judged sufficient for the lifetime of the system.*

STD-PRG4-TS.0013(1)	Prog_Limit(Segment#x) is decremented before each segment update.
Rationale	*In case the download procedure is interrupted, this ensures that the counter is decremented.*
Comment	*If Logical Block is contained into memory that needs erasing, Prog_Limit(Segment#x) is decremented before erasing. It is decremented after reprogramming in other cases.*

STD-PRG4-TS.0014(1)	If segment erasing is needed for reprogramming, Prog_Limit(Segment#01) (the first segment of the request download area) is decremented as part of executing BOOT_REQ_DOWNLOAD.
Rationale	*This gives a clear event for LOGICAL_BLOK_REPROG_CNTR(Logical Block ID) validation.*
Comment	

6.2.3 Internal states

State name	**Available states**	**Comment**
SECURITY_STATE	LOCK	In this state, it's not possible to modify the digest of a Logical Block or to change data contained into a Logical Block that matches with its digest.
	UNLOCK	In this state, there are no restrictions regarding the services available.
MEMORY_HDLR_STATE	OPEN	Boot-loader is ready to receive downloaded data and to store it into its memory.
	CLOSED	Boot-loader is not ready to receive downloaded data and to store it into its memory.

STD-PRG4-TS.0020(1)	SECURITY_STATE is set to LOCK when exiting the Programming session.
Rationale	*An ECU can not be left involuntarily to an UNLOCK state.*
Comment	*-*

STD-PRG4-TS.0021(1)	SECURITY_STATE is set to LOCK when on the Boot Initialization step.
Rationale	*-*
Comment	*Default value for SECURITY_STATE state is LOCK*

STD-PRG4-TS.0022(1)	MEMORY_HDLR_STATE is set to CLOSED when exiting the Programming session.
Rationale	*-*
Comment	*-*

STD-PRG4-TS.0023(1)	MEMORY_HDLR_STATE is set to CLOSED on the Boot Initialization step.
Rationale	*-*
Comment	*-*

STD-PRG4-TS.0024(1)	In case memory handler is contained into the Boot-loader, MEMORY_HDLR_STATE is set to OPEN on the Programming Session initialization.
Rationale	*-*
Comment	*-*

6.3 Boot-loader activity diagram

Boot-loader activity diagram is represented into Figure 2 - Boot-loader activity diagram part #1and Figure 3 - Boot-loader activity diagram part #2.

6.3.1 Entering Programming Session

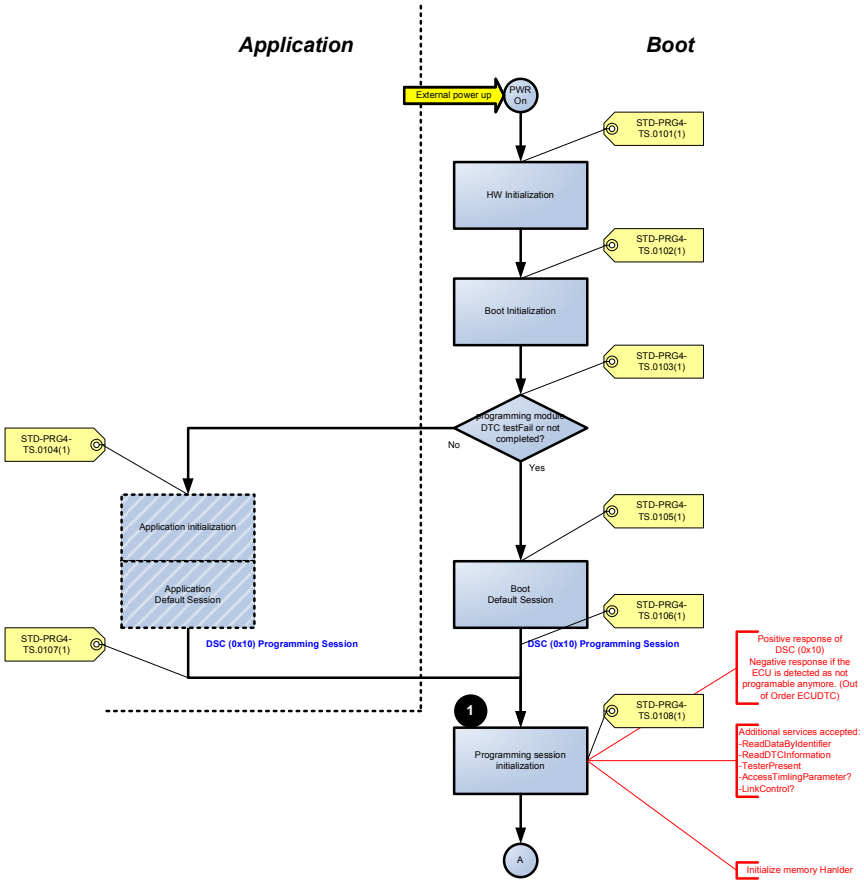


Figure 2 - Boot-loader activity diagram part #1

STD-PRG4-TS.0101(1)	After external ECU power up, Boot-loader starts its hardware initialization
Rationale	-
Comment	-

STD-PRG4-TS.0102(1)	After ECU's hardware initialization, the Boot-loader starts its software initialization. The Boot-loader initialization contains only the steps necessary to handle the subsequent decision step of starting application.
Rationale	-
Comment	-

STD-PRG4-TS.0103(1)	After ECU's software Boot-loader initialization, the Boot-loader checks the status of its DTCs
Rationale	-
Comment	-

STD-PRG4-TS.0104(1)	After Boot-loader DTC status check, if every DTC are tested and every tests returns no error, Boot-loader enable application running.
Rationale	*Checking DTC ensure that each Logical Block contains data that correspond to there associated digests*

Comment	*The mean of switching to the application is not described into this specification.*

STD-PRG4-TS.0105(1)	After Boot-loader DTC status check, if at least a DTC is not tested or indicate an error, the Boot-loader goes into its Default Session. Any remaining boot-loader initialization must be performed at this step.
Rationale	*If a DTC indicates that a test is not passed or returns an error, Boot-loader must not enable application running.*
Comment	*-*

STD-PRG4-TS.0106(1)	From Boot-loader default session, if Programming session is allowed, the positive response to successful BOOT_PROG_SESSION is sent just after it been entered into.
Rationale	*Positive response is sent when the service to go into programming session is effectively executed.*
Comment	*-*

STD-PRG4-TS.0107(1)	From the application, calling successfully the interface BOOT_PROG_SESSION exits application and enter the Boot-loader Programming Session initialization, the positive response to BOOT_PROG_SESSION is sent just after it been entered into.
Rationale	*-*
Comment	*-*

STD-PRG4-TS.0108(1)	
Rationale	
Comment	

6.3.2 Handling Programming procedures

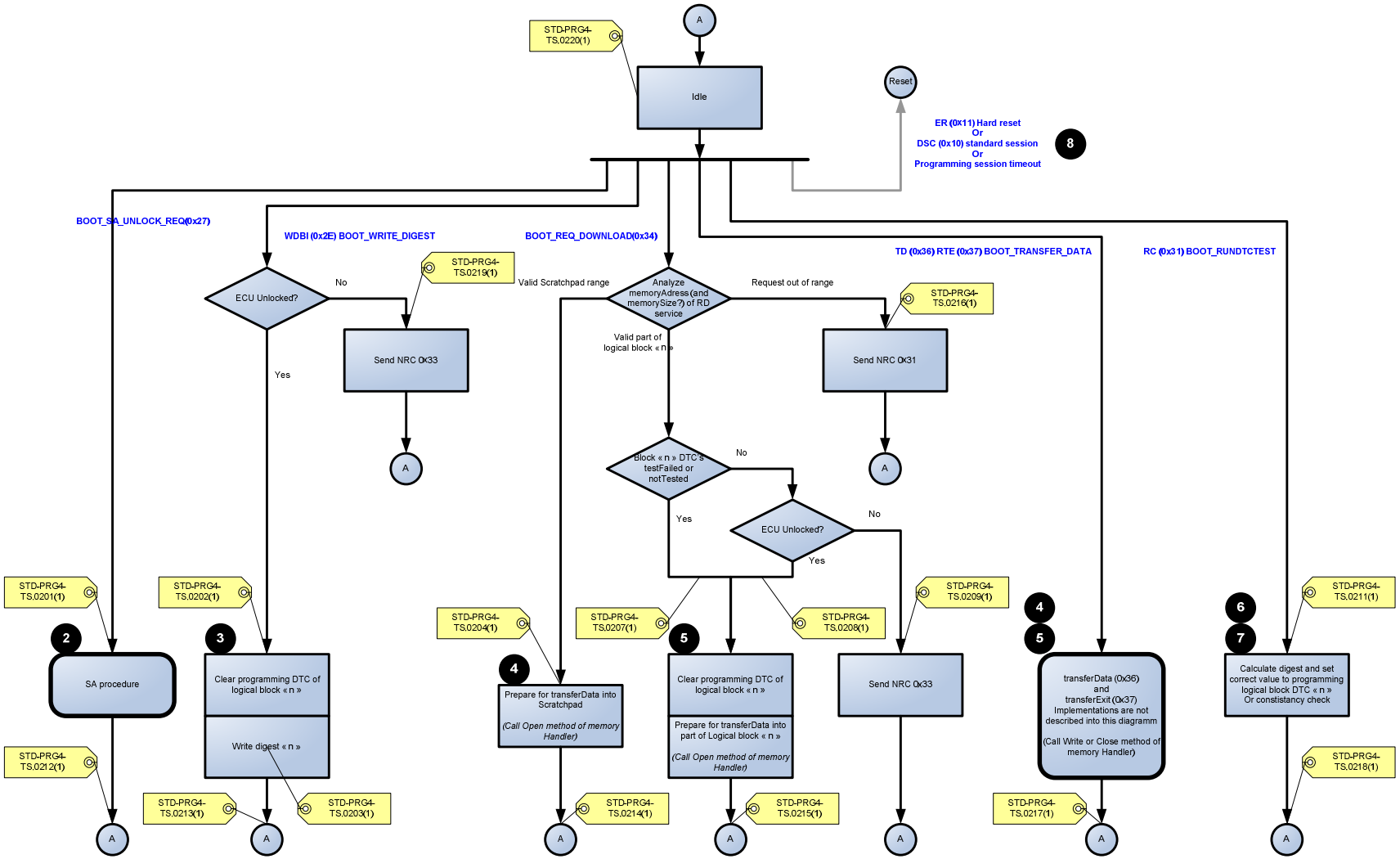


Figure 3 - Boot-loader activity diagram part #2

STD-PRG4-TS.0220(1)	
Rationale	
Comment	

STD-PRG4-TS.0201(1)	In idle state, calling the BOOT_SA_UNLOCK_REQ interface starts the unlocking security access procedure
Rationale	*-*
Comment	*Negative response can be sent here in case of impossibility to program ECU, e.g. programming counter is reached.*

STD-PRG4-TS.0212(1)	When security access procedure is over, whatever is the SECURITY_STATE, the Boot-loader goes back to its Idle states.
Rationale	*-*
Comment	*-*

STD-PRG4-TS.0202(1)	In idle state, calling the BOOT_WRITE_DIGEST forces the Boot-loader to check its SECURITY_STATE. In case it is into the state UNLOCK: • The associated DTC state is cleared, this mean that its states indicates that the test has not been passed. • The digest is written into ECU non volatile memory When all the above steps are accomplished, the positive response to BOOT_WRITE_DIGEST is returned.
Rationale	*-*
Comment	*-*

STD-PRG4-TS. 0203(1)	Each Logical Block is associated to a 16 bits DataIdentifier. Its datatRecord contains the Digest expected value. The value of the Dataidentifier is the 16 highest bits of the associated programmed/not programmed DTC.
Rationale	*This simplify communication between tool and Boot-loader*
Comment	*-*

STD-PRG4-TS.0212(1)	When writing digest procedure is over, the Boot-loader goes back to its Idle states.
Rationale	*-*
Comment	*-*

STD-PRG4-TS.0219(1)	In idle state, calling the BOOT_WRITE_DIGEST forces the Boot-loader to check its SECURITY_STATE. In case it is into the state LOCK, a negative response is sent as answer of BOOT_WRITE_DIGEST. Boot-loader turn then back to its Idle states.
Rationale	*-*
Comment	*-*

STD-PRG4-TS.0204(1)	This requirement addresses only Boot-loaders that support scratchpad downloading. In Idle state, calling BOOT_REQ_DOWNLOAD for a volatile memory download initialize memory handler. MEMORY_HDLR_STATE goes into OPEN state.
Rationale	*-*
Comment	*No security check is necessary for this step.*

STD-PRG4-TS.0214(1)	When preparing scratchpad download procedure is over, the Boot-loader goes back to its Idle states.
Rationale	
Comment	*-*

STD-PRG4-TS.0207(1)	In Idle state, if BOOT_REQ_DOWNLOAD is called for a non volatile memory

	download, Logical Block DTC's states are tested. In case the associated DTC is not tested or indicates an error: • The associated DTC state is cleared, this mean that its states indicates that the test has not been passed. • The memory handler is initialized accordingly. MEMORY_HDLR_STATE goes into OPEN state. When all the above steps are accomplished, the positive response to BOOT_REQ_DOWNLOAD is returned.
Rationale	*If an ECU Logical Block is operational (DTC indicates no error), it must be unlocked to be modified.* *If it's not operational, no unlocking is requested.* *This prevent unauthorized user to invalidate a Logical Block by modify its Digest.*
Comment	*Once Logical Block digest is written, the BOOT_REQ_DOWNLOAD can later be performed without any BOOT_SA_UNLOCK_REQ.*

STD-PRG4-TS.0208(1)	In Idle state, if BOOT_REQ_DOWNLOAD is called for a non volatile memory download, Logical Block DTC's states are tested. In case the associated DTC indicates that digest match with data and the SECURITY_STATE is into UNLOCK: • The associated DTC state is cleared, this mean that its states indicates that the test has not been passed. • The memory handler is initialized accordingly. MEMORY_HDLR_STATE goes into OPEN state. When all the above steps are accomplished, the positive response to BOOT_REQ_DOWNLOAD is returned.
Rationale	*If an ECU Logical Block is operational (DTC indicates no error), it must be unlocked to be modified.* *If it's not operational, no unlocking is requested.* *This prevent unauthorized user to invalidate a Logical Block by modify its Digest.*
Comment	*Once Logical Block digest is written, the BOOT_REQ_DOWNLOAD can later be performed without any BOOT_SA_UNLOCK_REQ.*

STD-PRG4-TS.0215(1)	When preparing download procedure is over, the Boot-loader goes back to its Idle states.
Rationale	*-*
Comment	*-*

STD-PRG4-TS.0209(1)	In Idle state, if BOOT_REQ_DOWNLOAD is called for a non volatile memory download, Logical Block DTC's states are tested. In case the associated DTC indicates that digest match with data and the SECURITY_STATE is into LOCK, a negative response is sent as answer of BOOT_REQ_DOWNLOAD. Boot-loader turn then back to its Idle states.
Rationale	*If an ECU Logical Block is operational (DTC indicates no error), it must be unlocked to be modified.* *If it's not operational, no unlocking is requested.* *This prevent unauthorized user to invalidate a Logical Block by modify its data.*
Comment	*-*

STD-PRG4-TS.0217(1)	When transfer data procedure is over, the Boot-loader goes back to its Idle states.
Rationale	*-*
Comment	*If transfer data was related to scratchpad, authenticity check can be performed. If it's not ok, transferred are erased.*

STD-PRG4-TS.0211(1)	In idle state, calling the BOOT_RUNDTCTEST forces Boot-loader to check if the associated Logical Block data matches with the corresponding digest.

	The corresponding DTC state is set according to the check result.
Rationale	*-*
Comment	*-*

STD-PRG4-TS.0218(1)	When autocontrol procedure is over, the Boot-loader goes back to its Idle states.
Rationale	*-*
Comment	*-*

If reprogrammable memory needs erasing before being written, this operation can be triggered on BOOT_REQ_DOWNLOAD.

6.4 Logical Block

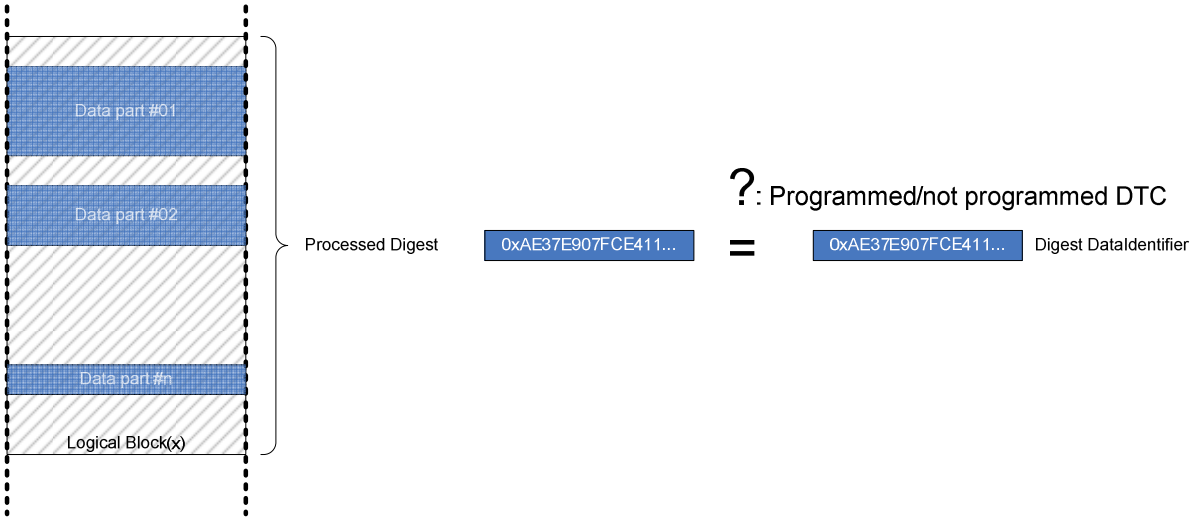


Figure 4 - Logical Block Digest

STD-PRG4-TS.0400(1)	Logical Block's segments address and size are statically defined.
Rationale	*-*
Comment	*-*

STD-PRG4-TS.0401(1)	Digest is computed on the entire Logical Block's area.
Rationale	*-*
Comment	*It's not mandatory that the programmed area covers the entire Logical Block.*

STD-PRG4-TS.0402(1)	Digest is computed using a standard cryptographic hash function.
Rationale	*Vulnerability of standard hash functions is publically known.*
Comment	*MD5 seems to be sufficient for this usage.*

STD-PRG4-TS.0403(1)	For a Logical Block, data that are not transferred using the BOOT_TRANSFER_DATA interface must be known by the tool.
Rationale	*ECU must ensure that the Digest can be also processed by the tool. This implies that for a Logical Block, bytes that haven't been transferred by BOOT_TRANSFER_DATA are all known by the tool.*
Comment	*This can be performed by an erasing of the entire Logical Block before being programmed.*

6.5 Scratchpad and Memory Handler.

STD-PRG4-TS.0500(1)	Default memory handler is loaded into scratchpad on Programming Session initialization.

Rationale	*-*
Comment	*-*

STD-PRG4-TS.0501(1)	If BOOT_REQ_DOWNLOAD memory address is not handled by memory handler contained into scratchpad, NRC 70 is returned to the tool.
Rationale	*-*
Comment	*-*

STD-PRG4-TS.0502(1)	ECU must check if signature of scratchpad match with authorized signature when close procedure of RAM Handler is called.
Rationale	*-*
Comment	*System supplier is responsible for providing signed memory handling routines.*

STD-PRG4-TS.0503(1)	If scratchpad signature does not match with authorized signature, default memory handler is reload into scratchpad.
Rationale	*-*
Comment	*-*

Physical Memory Handler interface:
- Open (param: memoryAddress and memorySize, return: ok or NRC)
- Write (transferRequestParameterRecord)
- Close.

Switch between implementations is performed on RD service, depending on memory address.

Interface implementations:
- Default Handler: resident in boot. Stub (returns NRC 22 on Open)
- Flash write capable routines (resident or contained in Scratchpad).
- RAM Handler: resident in boot, write into Scratchpad.